HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

**SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY**

A red logo with letters

Description automatically generated

**(IT3100E) OBJECT-ORIENTED PROGRAMMING**

**CAPSTONE PROJECT REPORT**

**TOPIC: VISUALIZATION OF OPERATIONS ON TREE DATA STRUCTURES**

**Group: 4**

**Class code: 147839**

**Supervisor: PhD. Tran The Hung**

**Members:**

Vo Thanh Vinh – Student ID: 20226074

Nguyen Vu Dung - Student ID: 20225964

Tran Duc Le Huy - Student ID: 20225978

Phan Dao Minh Quan - Student ID: 20225993

**Hanoi, June 16, 2024**

Table of Contents

[**MEMBER ASSIGNMENTS 3**](#_Toc169558141)

[**I. INTRODUCTION 4**](#_Toc169558142)

[**II. DESIGNS 4**](#_Toc169558143)

[1. Class Diagram 4](#_Toc169558144)

[2. Use-case Diagram 5](#_Toc169558145)

[**III. IMPLEMENTATION 5**](#_Toc169558146)

[1. Tree types 5](#_Toc169558147)

[a) Binary tree 5](#_Toc169558148)

[b) Generic tree 5](#_Toc169558149)

[c) Balanced tree 6](#_Toc169558150)

[d) Balanced binary tree 6](#_Toc169558151)

[2. Main operations 6](#_Toc169558152)

[a) Create a new tree 6](#_Toc169558153)

[b) Insert a node 6](#_Toc169558154)

[c) Delete a node 7](#_Toc169558155)

[d) Update the value of a node 7](#_Toc169558156)

[e) Traverse and Search for a node 8](#_Toc169558157)

[3. Bottom bar controls. 8](#_Toc169558158)

[a) Undo and Redo 8](#_Toc169558159)

[b) Pause and Continue 9](#_Toc169558160)

[c) Forward and Backward 9](#_Toc169558161)

[4. Searching algorithms 9](#_Toc169558162)

[a) Breadth-first Search (BFS) 9](#_Toc169558163)

[b) Depth-first Search (DFS) 9](#_Toc169558164)

[c) Iterative Deepening Depth-first Search (IDDFS) 10](#_Toc169558165)

[5. Operational animations 10](#_Toc169558166)

[**IV.** **CONCLUSION** 10](#_Toc169558167)

[**V.** **REFERENCES** 10](#_Toc169558168)

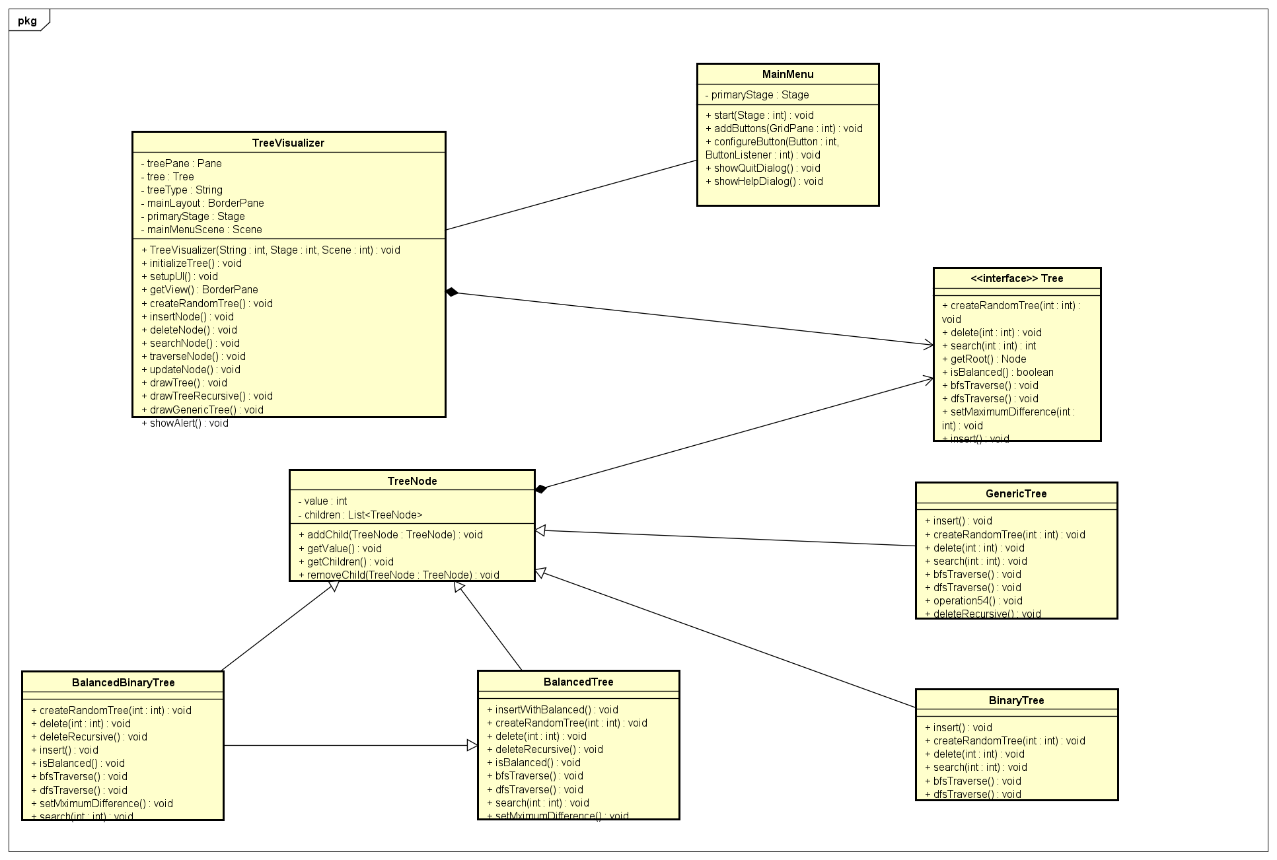
# **MEMBER ASSIGNMENTS**

|  |  |
| --- | --- |
| **Member name** | **Assignments** |
| Vo Thanh Vinh | * Implement the bottom bar controls: Undo, Redo, Pause, Continue, Forward, Backward. * Test and debug. * Write the report. |
| Nguyen Vu Dung | * Implement the basic tree types. * Design the Class and Use-case diagrams. * Test and debug. |
| Tran Duc Le Huy | * Implement the Main menu, the Graphical User Interface, and the Create, Insert, Delete and Update operations. * Test and debug. |
| Phan Dao Minh Quan | * Implement the Traverse and Search functions, the animations for all operations (except for the Create operation) and the traversal algorithms. * Test and debug. |

1. **INTRODUCTION:**

This report details the design and implementation of a program aimed at visualizing operations on different types of trees. The project includes Binary Tree, Generic Tree, Balanced Tree, and Balanced Binary Tree, each supporting various operations such as creation, insertion, deletion, updating, traversal, and search. The user interface includes a bottom bar with controls for undo, redo, pause, continue, forward, and backward, enhancing the interaction with the visualization process.

1. **DESIGNS:**
2. **Class Diagram:**

****

1. **Use-case Diagram:**

**A diagram of a company

Description automatically generated**

1. **IMPLEMENTATION:**
2. **Tree types:** Each of the tree types has its own Java class. We do have the “Tree” interface that acts as the core of the main menu and the “TreeNode” class for the operations on the tree types as well. The “TreeNode” class is the basic type for a tree node in this project, where the basic informational components are the value and the list of children, along with their get and set methods. Each tree type inherits the operations in the “TreeNode” class so that we can implement the operational methods on them and inherit those in the methods of the “TreeVisualizer” class later below. The operational methods are: get root, create a new tree, find the parent with the available slot for insertion, insert a new node, delete a node, search for a node, traverse, and clone the whole tree state for the bottom bar controls. Here are the short definitions and some properties of the tree types:
3. **Binary tree:**

A binary tree is a hierarchical structure in which each node has at most two children, commonly referred to as the left child and the right child. This structure is fundamental in computer science, allowing efficient searching and sorting operations.

1. **Generic tree:**

A generic tree, also known as an N-ary tree, is a more flexible structure where each node can have an arbitrary number of children. This type of tree is useful for representing hierarchical data that doesn't fit the strict binary constraints.

1. **Balanced tree:**

A balanced tree is a tree where each leaf node is “not more than a certain distance” away from the root than any other leaf.

1. **Balanced binary tree:**

A balanced binary tree has the properties of both a binary tree and a balanced tree.

1. **Main operations:** The main operations are implemented in the “TreeVisualizer” class. The “TreeVisualizer” class is the class for the graphical user interface of the program, it contains the operations and controls for when the operations are being performed. The operations here inherit the operational methods in the tree types, the “TreeNode” class and the “Tree” interface. The “drawTree” method is implemented in the “TreeVisualizer” class to help drawing and highlighting the tree on the screen for the operations below.
2. **Create a new tree:**

* For every types of tree, the system will ask the user to enter the number of nodes that he/she wants his/her tree to have. Since balanced tree and balanced binary tree types require the distance constraint, before asking for the number of nodes, the system will ask the user to enter the maximum distance from a leaf node to the tree’s root.
* When all conditions are satisfied, the system will generate a tree with random values and random edges.
* The method for this operation is “createRandomTree”. If the tree type that the user asks for does not require any distance constraints, the method will call for the “createRandomTreeWithoutMaxDifference” method, otherwise the “createRandomTreeWithMaxDifference” method will be called.

1. **Insert a node:**

* The Java method of this operation is the “insertNode” method, which inherits the “insert” methods of the tree types.
* If the tree’s root is null, the system will ask the user to enter the value for the root. Like the “Create” operation, if the tree type is balanced binary tree or balanced tree, the system will ask for the maximum distance between a leaf node and the root before asking the user to enter the root’s value.
* Otherwise, the system will ask for the parent value and the new value to be inserted. If the parent node does not exist or the new value to be inserted has already existed in the tree, the system will throw out a notification for the user. Balanced trees and balanced binary trees require the distance balancing constraint as well, so when the insertion of a new value violates the condition, the system will notify the user about the condition violence.
* If all conditions are satisfied, the “insertNode” method will traverse the tree to find the parent node, then assign the new value as a child of the searched parent node. The system will also draw the new node and its edge, then save the states of the tree (before and after insertion) for future undoes and redoes.

1. **Delete a node:**

* The method for this operation is the “deleteNode” method.
* When the operation is called, the system asks for the value to be deleted and the searching algorithm for it to search for the inputted value, then it calls for the “searchNodeForDeletion” method to utilize the chosen algorithm to search for the entered value. If the value to be deleted does not exist, the system will throw out a notification to the user about the invalid input. For balanced trees and balanced binary trees, the system will check whether the deletion affects the balancing property of the tree, and notify the user if it does.
* The system will also visualize the searching process, where each step is saved into the history list for future backward and forward controls (when the process is paused).
* If the entered value is found, the system will remove its node and all of its children out of the tree. For balanced trees and balanced binary trees, the system will also update the heights of the node to ensure the balancing property of the tree after the deletion.
* Once the deletion is completed, the system will save the current tree state for future undoes and redoes.

1. **Update the value of a node:**

* The system will ask the user for the value to be replaced and the new value. After the input, the system will traverse the tree to find the node to be update. If the value to be updated does not exist, or the new value has already existed in the tree, the system will give a message that the input is invalid. Otherwise, after the search, it will update the value as how the user wants.
* The system will also save the states of the tree (before and after the replacement) for future undoes and redoes.
* The method for this operation is the “updateNode” method.

1. **Traverse and Search for a node:**

* When this operation is called, the system asks the user to choose between 3 searching algorithms, then performs the visualization of the chosen algorithm on the tree.
* During the visualization flow, the searching steps are saved into a history list for future forward and backward controls (when the process is paused).
* The traverse flow can be paused then resumed after the last call of backward and forward controls occurs.
* The method for the traverse operation is the “traverseNode” method and that of the search operation is the “searchNode” method.

1. **Bottom bar controls:** Like the operations above, the controls are also implemented in the “TreeVisualizer” class. The controls are taken into action when an operation like traversing the tree or searching for a node is being performed, thus they affect the states of the tree at each step of the performing operation (Eg. which node and edge we are at now in this traversal step). For convenience when working with the tree states, we define a new type for the states called “TreeMemento”. The main components of this class are the tree, the list of highlighted nodes, and the set of permanently highlighted nodes, along with their get methods. This type is useful when we need to save or restore the current state of the tree for undo, redo, forward and backward controls.
2. **Undo and Redo:** These controls allow users to revert and reapply operations, providing flexibility in navigating through changes.

* The Undo control will maintain a stack type “TreeMemento” to save the operations, while the Redo control will maintain an another one to save the undone operations.
* When the Undo control is called, the system will pop the last operation in the undo stack and reverse the effect.
* When the Redo control is called, the system will pop the last undone operation in the redo stack and reapply it.

1. **Pause and Continue:** These controls manage the visualization's flow, allowing users to pause and resume operations.

* When the Pause control is called, the system will set a flag to indicate the paused state and temporarily halt the visualization to store the current state.
* When the Continue control is called, the system will clear the pause flag and resume the visualization from the paused state.

1. **Forward and Backward:** These controls enable step-by-step navigation through the operations, useful for debugging and educational purposes.

* These controls utilize a list as the history of the process of performing an operation, where each element is a tree state at each step of the process. They can only be called when the visualization of the process is paused.
* When the Backward control is called, the system will revert the visualization to the previous step of the operation and restore the previous state from the history list.
* When the Forward control is called, the system will advance the visualization to the next step of the operation and update the current state accordingly.
* The Continue control, if called, will resume the operation’s process at the step where the last call of either the Backward or Forward controls occurs.

1. **Searching algorithms:**
2. **Breadth-first Search (BFS):** The Breadth-first Search algorithm traverses nodes by level. It utilizes a queue to store the nodes in the traversal order. At first, the algorithm inserts the root node into the queue and marks it as visited, then it repeats the following sequence until the queue is empty: pops the node in front of the queue, then for each unvisited neighbor of the popped node, inserts the neighbor and mark it as visited. The algorithm terminates when all nodes of the tree has been traversed or an identified node is found. Its time and space complexity are both ,where is the number of nodes in the tree.
3. **Depth-first Search (DFS):** This algorithm traverses the nodes by their depths. It utilizes a stack to store the nodes in the traversal order. At first, the algorithm pushes the root node into the stack and marks it as visited, then it repeats the following sequence until the stack is empty: pops the node from the top of the stack, then for each unvisited neighbor of the popped node, pushes the neighbor and mark it as visited. The algorithm terminates when all nodes of the tree has been traversed or an identified node is found. Its time and space complexity are both ,where is the number of nodes in the tree.
4. **Iterative Deepening Depth-first Search (IDDFS):** This algorithm combines the space efficiency of the depth-first search algorithm and the optimality of the breadth-first search algorithm. It performs the depth-first search up to a certain limit, incrementally increasing the limit until all of the nodes are traversed or an identified node is found. It starts with the root node, where the depth limit is 0, then it performs the depth-first search up to the current limit, then increases the depth limit and performs the depth-first search until it reaches the goal.
5. **Operational animations:** To visualize the operations’ processes, we have initialized the shape components in the “TreeNode” class, which are the circles for nodes and a list of lines for the edges, and their Cartesian coordinates, along with their get and set methods for the “drawTree” method in the “TreeVisualizer” to utilize. We have also initialized a list of highlighted nodes, a set of permanently highlighted nodes and a set of highlighted edges in the “TreeVisualizer” class to highlight the nodes and the edges throughout the processes.
6. **CONCLUSION:**

This project effectively demonstrates the visualization of operations on various tree structures. By implementing Binary, Generic, Balanced, and Balanced Binary Trees and providing comprehensive operations, users gain an in-depth understanding of tree dynamics. The inclusion of bottom bar controls for undo, redo, pause, continue, forward, and backward enhances the learning experience, making the program an excellent educational tool for understanding tree operations in computer science.

1. **REFERENCES:**

* VisuAlgo of National University of Singapore.
* Other tutorials on YouTube.